DOI: 10.2298/CSIS111215015P

A Scale for Crawler Effectiveness on the Client-Side Hidden Web

Víctor M. Prieto[1], Manuel Álvarez[1], Rafael López-García[1], and Fidel Cacheda[1]

Comunication and Information Technologies Department - University of A Coruña
Campus de Elviña s/n - 15071 (A Coruña - Spain)
{victor.prieto, manuel.alvarez, rafael.lopez, fidel.cacheda}@udc.es

Abstract. The main goal of this study is to present a scale that classifies crawling systems according to their effectiveness in traversing the "client-side" Hidden Web.
First, we perform a thorough analysis of the different client-side technologies and the main features of the web pages in order to determine the basic steps of the aforementioned scale. Then, we define the scale by grouping basic scenarios in terms of several common features, and we propose some methods to evaluate the effectiveness of the crawlers according to the levels of the scale. Finally, we present a testing web site and we show the results of applying the aforementioned methods to the results obtained by some open-source and commercial crawlers that tried to traverse the pages.
Only a few crawlers achieve good results in treating client-side technologies. Regarding standalone crawlers, we highlight the open-source crawlers Heritrix and Nutch and the commercial crawler WebCopierPro, which is able to process very complex scenarios. With regard to the crawlers of the main search engines, only Google processes most of the scenarios we have proposed, while Yahoo! and Bing just deal with the basic ones.
There are not many studies that assess the capacity of the crawlers to deal with client-side technologies. Also, these studies consider fewer technologies, fewer crawlers and fewer combinations. Furthermore, to the best of our knowledge, our article provides the first scale for classifying crawlers from the point of view of the most important client-side technologies.

Keywords: Web Search, Crawlers, Hidden Web, Web Spam, JavaScript

1. Introduction

The World Wide Web (WWW) is the biggest information repository ever built. There are huge quantities of information that are publicly accessible, but as important as the information itself is being able to find, retrieve and gather the most relevant data according to users' needs at any given time.

Crawling systems are the programs that traverse the Web, following URLs to obtain the documents to be indexed by web search engines. From their origins, these systems have had to face a lot of difficulties when accessing human-oriented Web sites because some technologies are very hard to analyse: web

forms, pop-up menus, redirection techniques, mechanisms for maintaining user sessions, different layers which are shown or hidden depending on users' actions, etc. The pages that can only be accessed through these technologies constitute what we call Hidden Web [2] and, particularly, the set of pages that are "hidden" behind client-side technologies are called the "client-side Hidden Web". They represent a big challenge for programmers, since it is not easy to implement techniques that deal with them in a fully automatic way.

The objective of this paper is to present a scale for classifying crawlers according to their treatment of the client-side Hidden Web. This crawling scale will allows us a) to determine the capacities of each crawler dealing with client-side web technologies b) to know the percentage of the Web which has not been covered by crawlers and c) to know whether it is necessary to improve current crawling systems in order to deal with client-side web technologies in a better way.

The paper is organized as follows. In section 2 we analyse the related works. Section 3 shows the most important client-side technologies. In section 4 we list the difficulties that crawlers can find during their traversal and we classify them to create a scale. Section 5 shows this scale, which takes into account the level of use of each technology on the Web, the computational difficulty of each scenario and whether the crawlers have dealt with them. We also discuss some methods to assess the effectiveness of the crawlers at processing the aforementioned difficulties. In section 6 we show the website we have created to evaluate the crawlers and the results that each of them has obtained when traversing its pages. Finally, in sections 7 and 8 we explain the conclusions of the study and possible future works respectively.

2. Related works

There are many studies on the size of the Web and the characterisation of its content. However, there are not so many studies on classifying Web pages according to the difficulty for crawlers to process them. Weideman and Schwenke [20] published a study analysing the importance of JavaScript in the visibility of a Web site, concluding that most of the crawlers do not deal with it appropriately. However, according to the data submitted in [19] [4], currently the 90% of the Web pages use JavaScript.

From the point of view of crawling systems, there are many works aimed at creating programs that are capable of traversing the Hidden Web. There are some researches that tackle the challenges established by the server-side Hidden Web, by searching, modelling and querying web forms. We highlight HiWE [18] because it is one of the pioneer systems. Google [14] has also provided the techniques that it has used to access information through forms. Nevertheless, there are fewer studies about the client-side Hidden Web and they basically follow these approaches: they either access the content and links by means of interpreters that can execute scripts [16] [9], or they use mini-browsers such as the system proposed by Alvarez *et al.* [1].

There are also studies about the client-side technologies from the point of view of Web Spam. Their objective is to analyse the use of technologies in order to detect Web Spam techniques [10] such as Cloacking [21] [23], Redirection Spam [6] or Link Farm Spam [22].

Nevertheless, we do not know any scales that allow researchers to classify the effectiveness of the crawling systems according to their level of treatment of client-side Hidden Web technologies.

3. Client-side Web technologies

In order to define the scale, the first step is to understand the impact of the different client-side technologies on the Web ("Web Coverage"). For this purpose, we have used "The Stanford WebBase Project"[1] dataset, which is part of the "Stanford Digital Libraries Project"[2].

This dataset contains different kinds of sources and topics and more than 260 TB of data. For this analysis, we have created a 120 million page subset from the latest 2011 global datasets. In this dataset we have analysed technologies typically used to improve the user experience, such as:

- JavaScript [8], dialect of the ECMAScript standard, is an imperative and object-oriented language. It allows programmers to generate the interface dynamically.
- Applet, Java component of an application that is executed in a Web client.
- AJAX [11], technology that uses JavaScript for sending and receiving asynchronous requests and responses.
- VBScript [13], interpreted language created by Microsoft as a variant of Visual Basic.
- Flash [3], application that allows programmers to create vectorial interfaces.

Table 1 shows the results obtained by each technology (row) per month (column). According to these results, 60.30% of the documents contain JavaScript while only 2.58% contain ActionScript (Flash). The degree of occurrence of technologies like VBScript, Python and Tcl, is merely symbolic.

In short, these results confirm our assumption about the importance of knowing whether the crawlers treat the client-side technologies, since the use of JavaScript is generalized.

4. Occurrence of the main technologies

To define the basic levels of the scale we have identified the most commonly used mechanisms for generating links. To do this, we have relied on the technology analysis done in the "Client-side Web technologies" section and on a

[1] http://dbpubs.stanford.edu:8091/ testbed/doc2/WebBase/
[2] http://diglib.stanford.edu:8091/

Table 1. Analysis about Web technologies.

TECHNOLOGY	MONTH							TOTAL
	January '11	Febrary '11	March '11	April '11	May '11	June '11	July '11	
Pages	8519300	20057638	18700000	1230000	12677481	24505000	33453991	**119143410**
JavaScript	5791148	10436415	12672553	902780	6960075	16959179	18013475	**71735625**
	67.98%	**52.03%**	**67.77%**	**73.40%**	**54.90%**	**69.21%**	**53.85%**	**60.30%**
VBScript	7871	38243	19361	213	6253	20700	32529	**125170**
	0.09%	0.19%	0.10%	0.02%	0.05%	0.08%	0.10%	**0.11%**
Flash	257134	430772	584661	29376	273882	829380	669211	**3074416**
	3.02%	**2.15%**	**3.13%**	**2.39%**	**2.16%**	**3.38%**	**2.00%**	**2.58%**
Python	9	27	31	0	15	33	44	**159**
	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	**0.00%**
Tcl	228	42	228	3	26	1821	5083	**7219**
	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.02%	**0.01%**
Applets	139	2832	1049	5	2142	1176	2671	**10014**
	0.00%	0.01%	0.01%	0.00%	0.02%	0.00%	0.01%	**0.01%**

manual analysis of a subset of 1000 pages selected randomly from the dataset discussed above. We have also considered the Web design common practices [7] [12].

The following features have been identified:

- Text links, which constitute the lowest level of the scale.

```
<a href="a_110001001000000000_test...html">Luis
Ramirez Lucena</a>
```

- Simple navigations, generated with JavaScript, VBScript or ActionScript. This feature includes links that are generated by means of "document.write()" or similar functions in other languages, which allow designers to add new links to the HTML code dynamically.

```
<a href="JavaScript: ...">Paolo Boi</a>
```

- Navigations generated by means of an Applet.
- Navigations generated by means of Flash.
- In the case of Applets and Flash, we divide links into two types: those which are passed as an argument to the application and those whose URL is created as a string inside the code.
- Navigations generated by means of AJAX.
- Pop-up menus, generated by a script that is associated to any event.
- Links that are defined as strings in .java, .class files or any other kinds of text and binary files.
- Navigations generated in script functions. They can be written in any scripting language and the script can be either embedded inside the HTML or located inside an external file.
- Navigations generated by means of several kinds of redirections:

- Redirections specified in the `<meta>` tag.
- Redirections generated by the “onLoad” event of the `<body>` tag.
- Redirections generated by a script when an event in another page is fired (e.g.: the “onClick” event).
- Other redirections in script blocks, which are executed the moment the page is loaded.
- Redirections executed in an applet when the page is loaded.
- Flash redirections, executed when the page and its corresponding Flash file are processed.

In addition, the navigations that are generated with any of the methods we identified can create absolute or relative URL addresses. For the addresses that are generated by means of any scripting language, it is possible to recognize the following construction methods:

– A static string inside the Script.

```
menu_static_embedded_relative(){
document.location="a_10010101100000000...html";
}
```

– A string concatenation.

```
function menu_concatenated_embedded_relative(){
var out="";
out="a_1001001010000000000_test_menu" +
"_concatenated_embedded_relative.html";
document.location=out;
}
```

– Execution of a function that builds the URL in several steps.

```
function menu_special_function_embedded_relative(){
var a1="win",a2="dow",
a3=".location.",="replace",a5;
a5="('a_1001000110000000000_test_menu_special_function";
var a6="_embedded_relative.html')"; var i,url="";
for(i=1;i<=6;i++){
url+=eval("a"+i);
}
eval(url);
}
```

Furthermore, the distinct methods we have listed above can be combined. For example, some Web sites build pop-up menus dynamically, by means of “document.write()” functions. The number of possible combinations would be too high. Hence, this study only takes into account a reduced but significant subset, which is shown in Table 2.

The 70 scenarios we have proposed are combinations of the types of URLs, technologies, locations and methods of construction of URLs we identified, all of them shown in the columns of Table 2. At the bottom of the table we show the different types of redirections. The number of scenarios could be higher, but it would not provide more information about the use of client-side technologies on the Web or about the methods that crawlers use to discover links. For example, it is not necessary to take into account the combination of menus and "document.write()" because we can deduce the capacity of the crawler from the two base cases that have been included separately.

Table 2. Combination of the types of links.

Url Type	Technology	Location	String type		Test
Relative / Absolute	Text	Embedded	Static		1 - 2
	JavaScript	Embedded / External	Static / Concatenated / Special Function		3 - 38
	Document.write()	Embedded / External	Static / Concatenated / Special Function		3 - 38
	Menu JavaScript	Embedded / External	Static / Concatenated / Special Function		3 - 38
	Link in .java	External	Static		39 - 40
	Link in .class	External	Static		41 - 42
	Applet-Link HTML	Embedded	Static		43 - 44
	Applet-Link Class	External	Static		45 - 46
	Flash-Link HTML	Embedded	Static		47 - 48
	Flash-Link SWF	External	Static		49 - 50
	AJAX	Embedded	Static		61 - 62
	JavaScript with #	Embedded	Static / Special Function		63 - 66
	VBScript	Embedded	Static / Special Function		67 - 70
Relative / Absolute	Redirect	External	Static	Tag Meta	51 - 52
				Tag body	53 - 54
				JavaScript	55 - 56
				Applet	57 - 58
				Flash	59 - 60

5. The scale for web crawlers

To create the scale from the 70 scenarios we have proposed, we have implemented a grouping process based on their difficulty:

- Theoretical difficulty of processing each scenario.
- Practical difficulty (for existing crawling systems) to treat each scenario.

Regarding the first issue, we have performed a theoretical and an empirical analysis of the scenarios in order to determine the necessary modules for a crawler to process them. The theoretical analysis consists in calculating the processing modules that a crawling system would need to process each kind of difficulty ("Extraction Process" column in Table 3). For example, to treat the plain text scenario (level 1), we only need the crawler and a basic URL extractor. However, to extract URLs embedded in JavaScript code (level 2), we would also need a JavaScript interpreter, thus increasing the complexity.

After this, we have assessed our analysis by empirically processing the scenarios by means of well-known existing tools (JavaScript interpreters, Flash decompilers, and so on). Depending on the number and complexity of the modules that are needed to process a scenario, we have assigned a discrete score to it, which is shown in "Complexity" column of Table 3. The values are based on the difficulty of developing such modules and the computational expenses of their execution. Our grouping is based on the assumption that the scenarios which require a bigger number of modules are more difficult to treat.

To learn how crawling systems process the different scenarios, we have implemented a web site (see section 6.1) that contains all the scenarios shown in Table 2. Table 6 and Table 7 (see section 6) show the results obtained by the different crawlers that traversed the example website, grouped by technology and link building method. In this case, we have grouped the scenarios that have been processed by the same number of "crawling systems", because they are considered to have the same complexity.

Based on these analyses, we have clustered the pre-defined scenarios to form the 8 levels of the scale we propose, which is shown in Table 3. For each level there is a brief description, the scenarios that are part of it and their complexity. Table 3 also shows the frequency of appearance on the Web of each web technology that has been used in each scenario (column "F"). This frequency has been obtained from the results of the Web analysis we show in section 3 (Table 1). For example, the percentage of use of JavaScript is 60.30%, so, normalising to 1, the scenarios based on JavaScript will have a frequency of 0.6. In the case of text links, we have assumed that all the web pages have at least a text link, therefore we will use a frequency of 1. This information complements our scale and allows determining more accurately what portion of the Web is treated by each crawling system, as well as the maximum complexity level reached.

In the next section, we propose various evaluation methods for measuring the effectiveness of a crawler according to the scale.

5.1. Scale evaluation method

In order to evaluate a group of crawlers, one must set them up and run them on the testing site, creating a table similar to Table 6 to compare the results. Then we propose a list of evaluation methods in order to classify the crawlers according to the level of complexity of the "links" they have processed. As a

Table 3. Scale & Link classification by difficulty.

Level	Description	Scenarios	F	Complexity	Extraction Process
1	Text link	1,2	1	Very Low	Module of Crawling - Extractor of URLs
2	JavaScript/Document.Write/Menu - Static String - Embedded	3,4,15,16,27,28	0.6	Low	Module of Crawling - JS Interpreter
	JavaScript - Concatenated String - Embedded	6	0.6		Analyser of Redirects - Extractor of URLs
3	HTML/onBody/JavaScript Redirect	51,52,53,54,55,56	1	Low	Module of Crawling - JS Interpreter
	JavaScript # - Static String - Embedded	63,64	0.6		Extractor of URLs
4	VBScript - Static String - Embedded	67,68	0.01	Medium	Module of Crawling - Interpreter VBS
	VBScript - Special Function - Embedded	70	0.01		Extractor of URLs
5	JavaScript/Document.Write - Static String - External/Embedded	9,10,18	0.6	Medium /	Module of Crawling - Interpreter VBS
	Document.Write/Menu - Static String - External	21,22,33,34	0.6	High	Extractor of URLs
	Menu - Concatenated String - Embedded	30	0.6		
6	JavaScript/Document.Write/Menu-Concatenated String-External	12,24,36	0.6	Medium /	Module of Crawling - JS Interpreter
	Applet - Static String in HTML	43,44	0.03	High	Advanced extractor of URLs
7	JavaScript - Concatenated String - External/Embedded - Relative	5,11	0.6	High	Module of Crawling
	JavaScript - Special Function - External/Embedded - Relative	7,8,13,14	0.6		Advanced JS Interpreter
	Document.Write - Concatenated String - External/Embedded	17,23	0.6		Java decompiler
	Document.Write - Special Function - External/Embedded	19,20,25,26	0.6		Analyser of external files
	Menu - Concatenated String - External/Embedded - Relative	29,35	0.6		Advanced extractor of URLs
	Menu - Special Function - External/Embedded	31,32,37,38	0.6		
	Link in .java	39,40	0.03		
	AJAX Link - Absolute	62	0.6		
8	Link in .class	41,42	0.03	Very High	Module of Crawling
	Applet - Static String in .class	45,46	0.03		Advanced JS Interpreter
	Flash - Static String in HTML/SWF	47,48,49,50	0.3		Java decompiler - Flash decompiler
	Applet/Flash Redirect	57,58,59,60	1		Advanced VBS Interpreter
	AJAX Link - Relative	61	0.6		Analyser of external files
	JavaScript with # - Special Function - Embedded	65,66	0.6		Analyser of refirects
	VBScript - Special Function - Embedded	69	0.01		Advanced extractor of URLs

preliminary step, we specify the following concepts to help us formally define each method:

- Let be $S = \{Full\ set\ of\ scenarios\}$, being $|S|$ the total number of scenarios and S_i a binary value which establishes "1" if the crawler achieved the $i-th$ scenario of S or "0" in another case.
- Let be $N = \{Full\ set\ of\ levels\ that\ the\ crawler\ processes\ successfully\}$ being $|N|$ the total number of levels and $|N_i|$ each element of N.
- Let be $C = \{Full\ set\ of\ crawlers\}$, being $|C|$ the total number of crawlers, $C_i = \{Set\ of\ crawlers\ that\ achieved\ the\ scenario\ i\}$. The values of C_i are shown in Table 4. The set of 70 scenarios are shown in a matrix where the scenario ids are built taking the value of the upper row as the units, and the value of the left column as the tens.

Table 4. Values of C_i.

Scenarios		Units									
		0	1	2	3	4	5	6	7	8	9
	0	7	7	6	6	1	6	1	1	3	3
T	1	1	2	1	1	6	6	1	3	1	1
e	2	3	3	1	2	1	1	6	6	1	3
n	3	1	1	3	3	1	2	1	1	0	0
s	4	1	1	2	2	0	0	0	0	0	0
	5	5	5	5	5	5	5	0	0	0	0
	6	0	1	5	5	0	0	5	5	0	4

- Let be f_i the frequency (see Table 3) of occurrence of i.

We now describe each of the methods we propose:

- Simple Average: it treats the scenarios defined for each level in the same way, without taking into account their difficulty. It highlights the crawlers which treat the highest number of scenarios, and consequently pay more attention to the Hidden Web in general.

$$SA = \left(\frac{1}{|S|}\right) \sum_{i=1}^{|S|} S_i$$

- Maximum Level: this model sorts crawlers according to the highest level of difficulty they can process. A crawler obtains a score i if it has the capacity of processing the scenarios of that level and the levels below. Some crawlers process a certain level, but they cannot obtain pages from scenarios of a lower level. This could be due to some problems, like the low PageRank of a Web page and others. However, this evaluation method assumes that, if

a crawler is capable of dealing with a level i, it should be able to deal with lower ones.

$$ML = i \mid \forall i, j \in 1 \dots |N| \, N_i, N_j = 1 \Lambda i \geq j$$

– Weighted Average: this measure depends on the number of crawlers that have previously been able to process each scenario (C_i) and whether the new crawler has been able to process it (the binary value we have called S_i). Assuming that all crawlers can deal with most of the easiest scenarios, this method gives importance to the crawlers that can obtain the highest number of difficult resources in the client-side Hidden Web, since they are the only ones with can get points in the difficult scenarios.

$$WA = \left(\frac{1}{|C| + 1} \right) * \sum_{i=1}^{|S|} ((C_i + S_i) * S_i)$$

– Eight Levels: in this model each level has a value of one point. If a crawler processes all the scenarios of one level it obtains that point. For every scenario that the crawler processes successfully, it gets $1/n$ points, where n is the total number of scenarios that have been defined for that level.
Let $L = \{L_1 \dots L_8\}$ be the sets of scenarios which represent the 8 levels previously defined. Then L_{ij} is the $j-th$ scenario of $i-th$ level. The number of elements of each set can be different from the others.

$$\begin{array}{c} L_{11} \dots L_{1p} \rightarrow L_1 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ L_{81} \dots L_{8w} \rightarrow L_8 \end{array}$$

$$EL = \sum_{i=1}^{8} \sum_{j=1}^{|L_i|} \frac{L_{ij}}{|L_i|}$$

These models allow researchers to measure crawling systems capacity of obtaining links with a certain difficulty. However, in order to get a more conclusive opinion about which crawler processes the client-side Hidden Web the best, it is important to contextualise the technologies and their degree of occurrence in the Web. A crawler is not better than another if it processes a higher level in the scale when that level has few Web resources.

To solve this problem, we have defined the "Crawling Web Coverage" metric as the number of scenarios that a crawler solves, weighing up the technologies involved according to their frequency on the Web. We have also extended Weighted Average and Eight Levels, creating two new methods, called "Ranked Weighted Average" and "Ranked Eight Levels", which work like the basic methods but consider the degree of occurrence of each scenario on the Web.

To do this, we compare the results we have obtained in our study of Web technologies (section 3) and those obtained in the W3techs [19] and builtwith [4] web sites. The results are similar except for the frequent use of JavaScript.

According to their data 90% of the web pages use it, but in our study this number has decreased to 60%. We have chosen our result because their data are based on an analysis of pages included in the top 1 million or top 10,000 visits. We do not consider that those pages represent the reality of the Web, since they have high traffic and use innovative technologies to offer a better user experience.

The F (Frecuency) column of Table 3 shows the occurrence frequency of the base technology of each level. In this way, a crawler that processes a level with frequency 1 will be more valuable than one that processes a level with a lower frequency, since the probability of finding a link of the first type on the Web is much higher. On the other hand, a crawler that obtains links with a value of 0.01 will not get a high result, since this kind of links are scarce on the Web.

– Ranked Weighted Average

$$RWA = \left(\frac{1}{|C|+1}\right) * \sum_{i=1}^{|S|} ((C_i + S_i) * S_i * f_i)$$

– Ranked Eight Levels

$$REL = \sum_{i=1}^{8} \sum_{j=1}^{|L_i|} f_{i,j} * \frac{L_{i,j}}{|L_i|}$$

The Simple Average and Weighted Average methods and their ranked counterparts should be normalized in order to obtain values between 0 and 8, which make the methods easier to use and to compare.

As we can see, each method provides a different point of view on the capabilities of a crawler. Therefore, it is precisely this set of all methods that gives us a global vision of the processing capacity of the client-side technologies and the percentage of the web that is being treated.

6. Experimental results

In this section we describe the results of the experiments for the creation of the scale and its application in the classification of open-source and commercial crawlers. As a preliminary step, we describe the website we have designed to perform the tests.

6.1. Testing web site

In order to check how crawling systems deal with the different scenarios, we have created a Web site[3] for performing experiments. This Web site contains 70 links representing the 70 scenarios shown in Table 2. Those links use and combine the different technologies explained previously. With the purpose of

[3] http://www.tic.udc.es/∼mad/resources/projects/jstestingsite/

stimulating the indexing of the Web site, we have linked it from the main Web page of our institution Web site, we have written its content in English and we have added the biography of a famous chess player to each scenario. The latter measure was taken in order to avoid confusion with Web Spam sites.

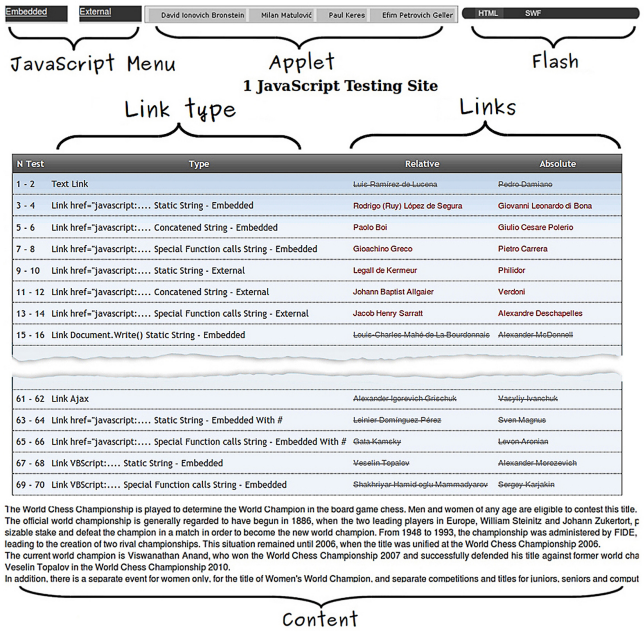

Fig. 1. Main page of the prototype.

Fig. 1 shows the main page of the prototype. It contains the following parts:

- At the top, from left to right, there are links in a JavaScript menu, links generated in an Applet and links in Flash.
- In the middle of the page, there is a table with four columns containing the test number, its type and the relative and absolute links respectively.
- At the bottom, below the table, it is the content.

Apart from that, we have created a page for the result of each test, so if a crawler can access certain contents, it will mean that it has been able to process the corresponding links. Fig. 2 shows a result page, consisting of the following elements:

- At the top-centre, it shows the test number and name.
- At the top-left it shows the reference code of the test, a binary mask that represents the features of the test.
- On the left side, it shows a table that enumerates the features of the test.
- The centre includes the biography of a chess grandmaster.

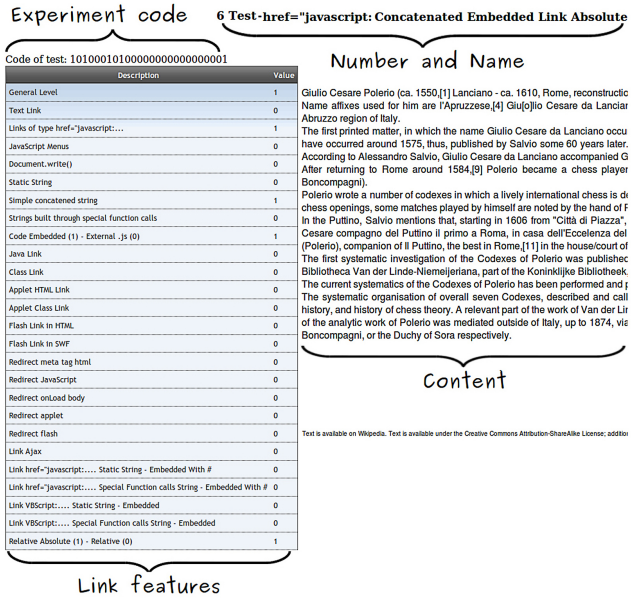

Fig. 2. Target Web page associated to the link of one scenario.

6.2. Experimental setup

For both the realization of the scale and the classification of crawling systems according to the evaluation methods, we have obtained two types of results.

The first one is based on the fact that a crawling system indexes the resources it obtains, and puts the data in a certain repository. Hence, we can extract information by querying this repository. On the other hand, we can extract access information from the log files of the Web server that maintains the testing Web site, in order to identify the crawlers that were accessing any page of that server. With this study we want to get further detail on how crawlers try to process each level. We have used the User-Agent and IP pairs published in the official robots page [4] to identify and remove non-independent crawlers, that is to say, those which depend on the crawler of a third party. In order to accelerate the process of visiting and indexing the site, we have registered the site in these crawlers: GoogleBot[5], Bing[6], Yahoo![7], PicSearch[8] and Gigablast[9].

For both the data in the logs and those obtained in the repositories, we have analysed the results of the crawlers of the major search engines as well as the

[4] http://www.robotstxt.org/
[5] http://www.google.es/addurl/
[6] http://www.bing.com/webmaster/SubmitSitePage.aspx
[7] http://siteexplorer.search.yahoo.com/submit
[8] http://www.picsearch.com/menu.cgi?item=FAQ
[9] http://www.gigablast.com/addurl

results of open-source and commercial ones. We have compared the global results in terms of the type of link and in terms of the method of construction of the URL string. First, we have studied each type of crawler independently, and then, comparing the results of both types.

These results have been used to evaluate each crawling system by means of the proposed methods and to obtain a final classification of each one in the scale we have proposed.

6.3. Open-source and commercial crawlers

We have analysed 23 open-source and commercial crawlers. Table 5 shows their most important characteristics from the point of view of the client-side technologies, along with the treatment of forms.

Among the features that they share, but we have not shown, we highlight: options about the use of a proxy, limitations in the number of documents, different protocols, inclusion/exclusion of file extensions, cookies, User-Agent, options about domains/directories, logging and some more.

Table 5. Open-source and Commercial Crawlers.

Crawler	Licence	JavaScript	Flash	Forms	Authentication	Thread
Advanced Site Crawler	Free	Yes	No	No	Yes	Yes
Essential Scanner	Free	No	No	No	Yes	No
Gsite Crawler	Free	No	No	No	No	Yes
Heritrix	Free	Yes	No	No	Yes	Yes
Htdig	Free	No	No	No	Yes	No
ItSucks	Free	No	No	No	Yes	Yes
Jcrawler	Free	No	No	No	No	No
Jspider	Free	No	No	No	Yes	Yes
Larbin	Free	No	No	No	Yes	Yes
MnogoSearch	Free	No	No	No	Yes	No
Nutch	Free	No	Yes	No	No	Yes
Open Web Spider	Free	No	No	No	No	Yes
Oss	Free	No	No	No	No	Yes
Pavuk	Free	Yes	No	Yes	Yes	Yes
Php Crawler	Free	No	No	No	No	No
WebHTTrack	Free	Yes	Yes	No	Yes	Yes
JOC Web Spider	Shareware	No	No	No	Yes	Yes
MnogoSearch	Shareware	No	No	No	Yes	Yes
Teleport Pro	Shareware	Yes	No	Yes	Yes	Yes
Visual Web Spider	Shareware	No	No	No	Yes	Yes
Web Data Extractor	Shareware	No	No	No	Yes	Yes
Web2Disk	Shareware	Yes	No	Yes	Yes	Yes
Web Copier Pro	Shareware	Yes	Yes	Yes	Yes	Yes

After examining these and other features, and taking into account the evaluation of their results in the treatment of client side technologies, we have selected the seven best crawlers among those shown in Table 5.

Among the open-source crawlers, we have chosen Nutch[10] [5], Heritrix[11] [15], Pavuk[12] and WebHTTrack[13], and among the commercial crawlers we have chosen Teleport[14], Web2disk[15] and WebCopierPro[16].

6.4. Results for open-source and commercial crawlers

First, we have studied the results according to the content of the Web site that has been indexed by the different open-source and commercial crawlers. To that end, we have analysed the repositories that the crawler generated during its execution. The crawler that achieves the best results is WebCopierPro (left side in Table 6), which processed 57.14% of the levels, followed by Heritrix with 47.14% and Web2Disk with 34.29%. Only a few of them get values beyond 25% in most types of links, and even those are unable to get links in many cases.

It is also important to notice the poor results that they have obtained for redirections, especially in the case of WebCopierPro, which has not been able to deal with any of them, although it has got results of 100% in harder levels. None of the crawlers has reached the 100% in redirections, because they have not been able to process pages with redirections embedded in Applets or Flash. Although they have downloaded the pages, they have not executed the Flash or the Applet that generates the redirection.

If we analyse the results by type of link, which are shown in the left columns of Fig. 3 (section 6), we will obtain a vision about which types of links are processed by a bigger number of crawlers. Apart from the 100% obtained for text links, we can verify that they can complete 35% to 40% of the scenarios of href="javascript..."; "document.write()"; menu links; links with "#" and VBScript.

They only achieve 7% of the links in .class or .java files and those which have been generated with AJAX. None of them has got links from Flash, but this can be due to the scarce attention that crawlers pay to these kinds of links.

The last rows of Table 6 classify the links by their construction method and show how many of them have been processed by each crawler. If we calculate the mean of the percentages, we have 42.52% for static links, 27.38% for links generated by string concatenation and 15.18% for links that generated by means of functions. This indicates that most of the crawlers search URLs by means of regular expressions.

[10] http://nutch.apache.org/
[11] http://www.archive.org
[12] http://www.pavuk.org
[13] http://www.httrack.com
[14] http://www.tenmax.com/teleport/pro/home.htm
[15] http://www.inspyder.com/ products/Web2Disk/Default.aspx
[16] http://www. maximumsoft.com/products/wc_pro/overview.html

Table 6. Summary of results for open-source and Commercial crawlers. On the left side in the repositories and on the right side in the logs.

REPOSITORIES (Passed — %)														LOGS (Passed — %)					
	Heritrix		Nutch		Pavuk		Teleport		Web2Disk		WebCopierPro		WebHTTrack	Teleport		WebHTTrack	
Text link	2	100.00%	2	100.00%	2	100.00%	2	100.00%	2	100.00%	2	100.00%	2	100.00%	2	100.00%	2	100.00%
Href=" JavaScript link	6	50.00%	3	25.00%	0	0.00%	3	25.00%	5	41.67%	12	100.00%	3	25.00%	4	33.33%	3	25.00%
Document.write link	6	50.00%	3	25.00%	0	0.00%	2	16.67%	4	33.33%	12	100.00%	2	16.67%	3	25.00%	2	16.67%
Menu link	6	50.00%	3	25.00%	0	0.00%	2	16.67%	4	33.33%	12	100.00%	2	16.67%	3	25.00%	2	16.67%
Flash link	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	2	50.00%
Applet link	2	50.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	2	50.00%	0	0.00%	2	50.00%
Redirects	6	60.00%	6	60.00%	2	20.00%	6	60.00%	4	33.33%	0	0.00%	6	60.00%	6	60.00%	6	60.00%
Class or Java link	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	2	50.00%	0	0.00%	0	0.00%	0	0.00%
AJAX link	0	0.00%	1	50.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Links with #	2	50.00%	2	50.00%	0	0.00%	2	50.00%	2	50.00%	0	0.00%	2	50.00%	3	75.00%	2	50.00%
VBScript link	3	75.00%	3	75.00%	0	0.00%	3	75.00%	3	75.00%	0	0.00%	2	50.00%	3	75.00%	2	50.00%
Static string link	26	61.90%	19	45.24%	4	9.52%	18	42.86%	22	52.38%	16	38.10%	20	47.62%	18	42.86%	22	52.38%
Concatenated string link	6	50.00%	2	16.67%	0	0.00%	1	8.33%	1	8.33%	12	100.00%	1	8.33%	1	8.33%	1	8.33%
Special function string link	1	6.25%	2	12.50%	0	0.00%	1	6.25%	1	6.25%	12	75.00%	0	0.00%	5	31.24%	0	0.00%
Tests passed	33	47.14%	23	32.86%	4	5.71%	20	28.57%	40	57.14%	40	57.14%	21	30.00%	24	34.29%	23	32.86%

From these results, we conclude that the probability of finding links by means of regular expressions or treatment of text is inversely proportional to the difficulty of generation of the link.

Summarizing the data of Table 6, we conclude that only one third of the links that are generated with client-side technologies are treated by open-source or commercial crawlers.

As a complement to these results, we have analysed the entries that each crawler has generated in the logs of our Web server. The right side of Table 6 shows the results of the crawlers in the logs for the cases where these are different from the results of analysing the repository (left side). Those differences are due to:

– WebHTTrack has tried to access Flash scenarios, but it has not taken into account that the structure to pass parameters to Flash is "parameter1=data1& parameter2=data2&..." so it considers the whole string as an URL although it is not.

```
"GET mad/resources/projects/jstestingsite/1_test/
link_relative=a_10000100100001000000000000
_test_flash_link_html_relative.html&link_absolute=
http://www.tic.udc.es/~mad/resources/projects/
jstestingsite/1_test/a_100001001000010000000000001
_test_flash_link_html_absolute.html"
```

– Teleport has tried to access absolute URLs that have been generated by a function. It detected "http..." patterns and searched the corresponding ".html" patterns that indicate the end of the address. Nevertheless, it has not detected that some variables have been defined in the middle, so the URL it has generated is incorrect.

```
"GET /~mad/resources/projects/jstestingsite/1_test/";
var%20a="a_10001001100000000000000001
_test_document_write_function_special_embedded_absolute.html"
```

6.5. Results for crawlers of the main web search engines

In order to perform the analysis of the crawlers that the main Web search engines use, we registered the Web site on the 23rd of May of 2011 in the different search engines. Once it was available, we waited for 70 more days. Then, we checked the logs of the Web site in order to know which crawlers had accessed at least the main page of the Web site. Table 7 contains the results we obtained. It does not show the results for Alexa, Ask, Bing, Gigablast and PicSearch, since they have not indexed the testing Web site. Only Google and Yahoo! indexed it. Among the factors that could induce some of the crawlers to skip the Web site, we could remark the following:

– Low PageRank [17].

Table 7. Summary of the results for crawlers of the main Web search engines.

	Google (Passed — %)		Yahoo! (Passed — %)	
Text link	2	10000%	1	50.00%
Href="javaScript link	**6**	**50.00%**	0	0.00%
Document.write link	**6**	**50.00%**	0	0.00%
Menu link	**6**	**50.00%**	0	0.00%
Flash link	0	0.00%	0	0.00%
Applet link	0	0.00%	0	0.00%
Redirects	**6**	**50.00%**	2	20.00%
Class or java link	0	0.00%	0	0.00%
AJAX link	0	0.00%	0	0.00%
Links with #	**4**	**100.00%**	0	0.00%
VBScript link	**1**	**25.00%**	0	0.00%
Static string link	**17**	**40.48%**	3	7.14%
Concatenated string link	**6**	**50.00%**	0	0.00%
Special function string link	**8**	**50.00%**	0	0.00%
Total passed	**31**	**44.29%**	3	4.29

- It is stored in a deep level inside the Web site of the department.
- High content of code. This can make crawlers' heuristics decide that the Web site might contain Malware or links too difficult to analyse.

Google processed 44.29% of the links. This implies that neither the design nor the content nor the other variables inhibited crawlers to try to traverse the Web site. Yahoo! also processed some links, but only the easiest ones.

GoogleBot processed 50% of many of the levels we proposed. The other half was not processed because the code was stored in an external file that the crawler did not analyse. If these files were analysed, we suspect that GoogleBot would achieve much better results. The links that GoogleBot did not process included technologies like Flash, Applets and AJAX or file types like .class and .java. These crawlers have shown the same results in the indices (querying their search engine) and in the logs of our Web server. Hence, we do not show the results separately.

6.6. Result comparison

Looking at the overall results (last row) of Tables 6 and Table 7, we see that generally, open-source and commercial crawlers achieve better results. Only Google, getting processed 34 of the 70 scenarios, has similar performance.

This can be due to the fact that an open-source or a commercial crawler can be configured by the user, who establishes what kind of links should be followed without taking into account the performance or the security. These features cannot be ignored by the crawlers of the main Web search engines.

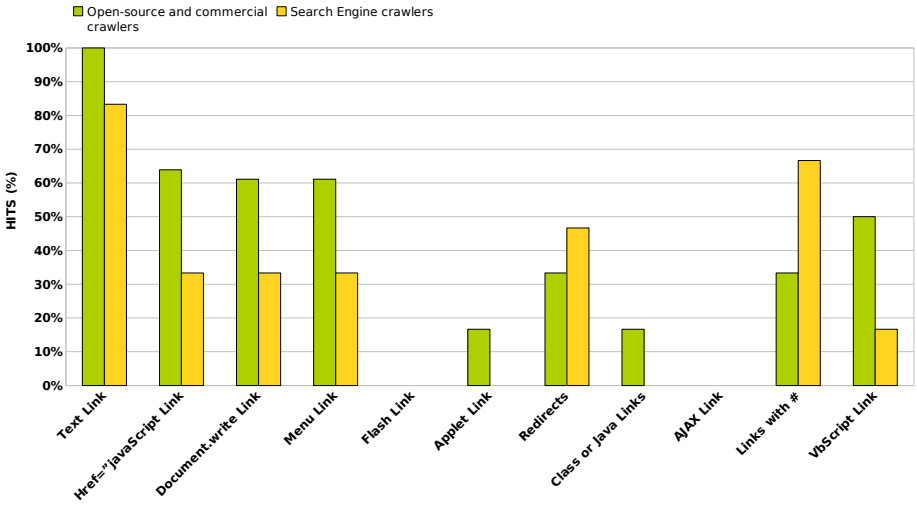

Fig. 3. Comparison among crawlers by link type.

Fig. 3 compares the results by technology used in the links. Once again, open-source crawlers have been more effective. One thing we would like to highlight is that the curve that every group of crawlers draw is similar. Hence, we can conclude that despite achieving a lower number of links in each technology, crawlers show the same interest in processing the same kind of technologies.

6.7. Classification of the crawlers according to the scale

After obtaining the results of each crawler in the website, we evaluate the crawling systems in the scale, using the evaluation methods we have proposed. Fig. 4 shows these results.

- For Simple Average, WebCopier gets the best results, followed by Heritrix and Google.
- For Maximum Level, Google places first since it processes level 8 links. It is followed by WebCopier, which obtains 7 points, and Heritrix, which obtains 6 points. As Google achieves the maximum level in this model but not in others, we can conclude that it has the capacity to deal with every scenario, but it does not try some of them because of internal policies.
- For the Weighted Average measure, WebCopier is followed by Google, Heritrix and Nutch.
- For Eight Levels, top places are for Heritrix, Web2Disk and WebCopier. GoogleBot places fourth. This means that the three top crawlers have dealt with a big quantity of levels in each group or they have gone through links that were part of a group with few links, which makes each link more valuable.

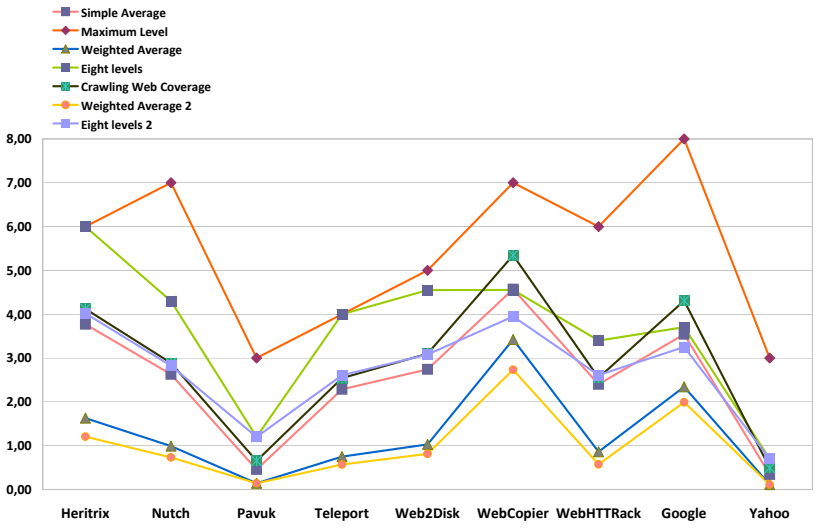

Fig. 4. Results according to the proposed scales.

However, the frequency of the technologies in the Web should be taken into account too. That is why we also show the Crawling Web Coverage metric and the results of the Ranked Weighted Average and Ranked Eight Levels methods, which are based on such metric. WebCopier, Google and Heritrix get the best results, so they are the ones that deal with the biggest portion of the Web.

- For Ranked Weighted Average, crawlers obtain lower results, but quite similar to the previous ones. Once again, WebCopier and Google are at the top at a certain distance from the rest.
- For Ranked Eight Levels, almost all crawlers decrease their performance between 1 and 2 points.

When a crawler gets a high score in the previous assessments and a low one in these, it means that it has been able to traverse difficult links, but that the technologies these links are made of do not have a significant number of occurrences on the Web.

Weighted results evaluate if a crawler can analyse client-side technologies on Web sites, but they also analyse if those capabilities are appropriate for treating the current state of the Web. We conclude that the best crawlers in both quantity and quality are Google and WebCopier, followed by Heritrix, Nutch and Web2Disk. It is important to highlight the results of GoogleBot. It takes into account a wide range of technologies although it is oriented to traverse the entire Web and it makes a lot of performance and security considerations.

7. Conclusions

This article proposes a scale that allows us to classify crawling systems according to their effectiveness when they access the client-side Hidden Web. It takes into account a global study on 120 million pages to determine the most commonly used client-side technologies and another one to explain the specific methods to generate links. It also includes the crawlers of the main search engines, as well as several open-source and commercial ones. Hence, we believe it is more exhaustive than the work by M. Weideman and F. Schwenke [20].

In order to classify the distinct crawling systems, we have created a Web site implementing all the difficulties that we had included in the scale.

The results we have obtained show that, for both the levels that the crawlers have tried and the levels they have achieved, most of the times they try to discover new URLs processing the code as text and using regular expressions. It is true that this allows them to discover a big amount of scenarios and that the computational expenses are not high, but we conclude that most of the URLs which are located inside client-side technologies are not discovered. The only crawlers that achieve good results at processing the client-side Hidden Web are WebCopier and GoogleBot. They surely are using an interpreter that allows them to execute code.

8. Future works

One of the future works is the design, implementation and testing of an algorithm that is capable of dealing with all the levels proposed in the scale. The algorithm should be composed of different modules for each level, starting with an analysis based on text and regular expressions, and finishing with the use of interpreters and mini-browsers. As a consquence, each module would have more computational cost, but they should be executed only if there are evidences pointing to the possible finding of new URLs in web pages that are not Web Spam. The tests should measure both the efficacy and the efficiency of the algorithm, since it would be designed to be included in a global crawling system.

A long term work would be modelling and implementing a high-performance crawler that includes the aforementioned algorithm and the Web Spam detection module. With both features we would have a high-performing safe crawler which would grant access to the hidden web content that, as we proved in this article, is not completely processed by crawlers.

References

1. Álvarez, M., Pan, A., Raposo, J., Hidalgo, J.: Crawling Web Pages with Support for Client-Side Dynamism. In: Yu, J., Kitsuregawa, M., Leong, H. (eds.) Advances in Web-Age Information Management, Lecture Notes in Computer Science, vol. 4016, chap. 22, pp. 252–262. Springer Berlin / Heidelberg, Berlin, Heidelberg (2006)

2. Bergman, M.K.: The Deep Web: Surfacing Hidden Value. Journal of Electronic Publishing 7(1) (Aug 2001)
3. Braunstein, R., Wright, M.H., Noble, J.J.: ActionScript 3.0 Bible. Wiley (Oct 2007)
4. BuiltWith: Web Technology Usage Statistics. http://trends.builtwith.com/ (2011)
5. Cafarella, M., Cutting, D.: Building Nutch: Open Source Search: A case study in writing an open source search engine. ACM Queue 2(2) (2004)
6. Chellapilla, K., Maykov, A.: A taxonomy of javascript redirection spam. In: Proceedings of the 3rd international workshop on Adversarial information retrieval on the web. pp. 81–88. AIRWeb '07, ACM, New York, NY, USA (2007)
7. Danielson, D.R.: Web navigation and the behavioral effects of constantly visible site maps. Interacting with Computers 14(5), 601–618 (2002)
8. Flanagan, D.: JavaScript: The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3rd edn. (1998)
9. Google: V8 JavaScript Engine. http://code.google.com/p/v8/ (2011)
10. Gyongyi, Z., Molina, H.G.: Web Spam Taxonomy. In: First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb 2005) (2005)
11. Holdener, III, A.T.: Ajax: the definitive guide. O'Reilly, first edn. (2008)
12. Kalbach, J., Bosenick, T.: Web page layout: A comparison between left- and right-justified site navigation menus. J. Digit. Inf. 4(1) (2003)
13. Kingsley-Hughes, A., Kingsley-Hughes, K., Read, D.: VBScript Programmer's Reference. Wrox Press Ltd., Birmingham, UK, UK, 3rd edn. (2007)
14. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's deep web crawl. Proc. VLDB Endow. 1, 1241–1252 (August 2008)
15. Mohr, G., Kimpton, M., Stack, M., Ranitovic, I.: Introduction to heritrix, an archival quality web crawler. In: 4th International Web Archiving Workshop (IWAW04) (2004)
16. Mozilla: Mozilla rhino javascript engine (2011), http://www.mozilla.org/rhino/
17. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project (1998)
18. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: Proceedings of the 27th International Conference on Very Large Data Bases. pp. 129–138. VLDB '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
19. W3Techs: World Wide Web Technology Surveys. http://w3techs.com/ (2011)
20. Weideman, M., Schwenke, F.: The influence that JavaScript has on the visibility of a Website to search engines. Information Research 11(4) (Jul 2006)
21. Wu, B., Davison, B.D.: Cloaking and redirection: A preliminary study. In: AIRWeb '05: Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (May 2005)
22. Wu, B., Davison, B.D.: Identifying link farm spam pages. In: Special interest tracks and posters of the 14th international conference on World Wide Web. pp. 820–829. WWW '05, ACM, New York, NY, USA (2005)
23. Wu, B., Davison, B.D.: Detecting semantic cloaking on the web. In: Proceedings of the 15th International World Wide Web Conference. pp. 819–828. ACM Press (2006)

Víctor M. Prieto graduated in Computing Science at the University of A Coruña in September of 2007 and he is a Ph.D. candidate in Computer Science at the same university. At this moment, he is part of the Telematics Group of the Department of Information and Comunications Technologies at the same university. His main research fields are web crawling, hidden web and web spam.

Manuel Álvarez is an Associate Professor in the Department of Information and Communication Technologies, at the University of A Coruña (Spain). He received his Bachelor's Degree in Computer Engineering from the University of A Coruña in 1999 and a Ph.D. Degree in Computer Science from the same University in 2007. His research interests are related to data extraction and integration, semantic and Hidden Web. Manuel has managed several projects at national and regional level in the field of data integration and Hidden Web accessing. He has also published several papers in international journals and has participated in multiple international conferences. He also teaches a Master's degree at the University of A Coruña and is consultant for Denodo Technologies (a telematic engineering company).

Rafael López-García got his Degree in Computing Science at the University of A Coruña in September of 2006. In the years that followed, he has focused on teaching and research, studying the postdegree at the same university. His primordial fields of interest in Computing Science are web technologies and distributed systems in Information Retrieval (IR), field in which he wants to write his Ph.D. Thesis.

Fidel Cacheda is an Associate Professor in the Department of Information and Communications Technologies at the University of A Coruña (Spain). He received his Ph.D. and B.S. degrees in Computer Science from the University of A Coruña, in 2002 and 1996, respectively. He has been part of the Department of Information and Communication Technologies, the University of A Coruña, Spain, since 1998. From 1997 to 1998, he was an assistant in the Department of Computer Science of Alfonso X El Sabio University, Madrid, Spain. He has been involved in several research projects related to Web information retrieval and multimedia real time systems. His research interests include Web information retrieval and distributed architectures in information retrieval. He has published several papers in international journals and has participated in multiple international conferences.

Received: December 15, 2011; Accepted: April 9, 2012.